QUERY PREDICTION IN VERY LARGE DATABASE SYSTEMS

A THESIS

SUBMITTED TO THE FACULTY OF CLARK ATLANTA UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE

BY

JANICE OLIVIA BOLDS

DEPARTMENT OF COMPUTER SCIENCE

ATLANTA, GEORGIA

JULY 1990

ABSTRACT

COMPUTER SCIENCE

BOLDS, JANICE O. B.S. MEMPHIS STATE UNIVERSITY, 1987

QUERY PREDICTION IN VERY LARGE DATABASE SYSTEMS

Advisor: Dr. Kofi Apenyo

Thesis dated July, 1990

A major area of concern with very large databases is that of query and access time. This area has played a major role in the design and development of current large database systems. Due to the volume of data stored in these systems, retrieval of data can turn out to be very time consuming. While fast response times are not vital to all database systems, those systems involved in real-time applications require immediate results.

This study will focus upon the development of a system for anticipating queries in large scientific databases. The methodology developed in this study offers a system that will not only respond to a given query, but also to potential queries. Based upon a given user query and the database schema, a pattern of likely queries will be predicted. Thus the method developed in this study offers a powerful tool for database environments requiring real-time applications since responses to anticipated queries will be readily available.

Acknowledgements

My deepest appreciation is extended to Dr. Kofi Apenyo for the continuous support and guidance he provided during this project. Also a special thanks to my family and friends who continually encouraged and inspired me.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

The impact of computer technology is receiving ever increasing attention. Computers have become a vital part of our life-style and have imposed drastic changes upon our society.

Traditional services and jobs have been lost to computer systems that perform routine tasks fast, efficiently and reliably. Consequently, today's computer systems have grown from mere number crunchers into sophisticated information utilities that are capable of storing, retrieving, updating, deleting and distributing huge quantities of data. Systems composed of large collections of integrated files, or very large databases have presented an answer for our societies seemingly unquenchable thirst for information. It appears as though the more information we are able to obtain, the more need for additional information is found. This quest for knowledge has played a major role in increasing the popularity of very large databases and mandating these systems as a necessity in business, industry, and government.

Because record-keeping and decision-making in organizations are increasingly based on information stored in database systems, the need for improved

hardware and software technology for implementing, managing and utilizing these databases has emerged into a primary area of interest. This need is particularly crucial in such a heterogeneous society where the abundance and assortment of interrelationships need to be identified, represented, explored, explained and optimized. As the information covering the array of interrelationships within our society expands, the urgency for large scale systems becomes more apparent.

Characteristics of Very Large Databases

There are several properties a system might contain in order to be considered a very large database. Some characteristics of these systems are:

- provides diversified services to many users at several sites.
- can be viewed as large collections of data combined with software programs which process transactions in order to retrieve, update, insert and delete data.
- assembled with the aid of commercially available database management system packages such as Ingres, Oracle, Sybase and DB2.
- systems occupy hundreds and even thousands of megabytes of storage.
- contain large collections of related tables containing data and definitions of database objects.

Very large databases essentially possess many of the characteristics found in mundane databases. However, the major distinction is found in the volume of data collected and stored in these systems. Yet due to the vagueness of the term "very large," the general size is not very clear.

Various authors have characterized very large database systems. The following are examples of how some authors have classified these systems.

Lockemann and Neuhold[1] consider a very large database as a system containing "up to several hundred million characters." The authors further state "The emerging development of large-scale storage devices (10^{12} - 10^{15} bits) suggests that a large share of future computer systems will be oriented towards the management of very large data bases." The authors focus their characterization of very large databases in terms of the number of characters the system will hold. Lockemann and Neuhold also suggest that very large databases contain a minimum storage capacity in the range of gigabytes.

Larger database systems are depicted by Inmon [2] as accommodating "a voluminous amount of a few types of data or [it] may contain a large and varied collection of data types which when put together form a massive amount of data." Inmon designates two ways in which a database is classified as a large-scale system.

In the author's perception, a large amount of data contained in a few tables constitutes a very large system. On the other hand, a database composed of several tables when combined yield enormous amounts of data designates a very large system as well.

Fleming and Von Hale[3] consider a very large database to be a system in which "at least one table contains several million rows, or where at least one table contains multiple gigabytes of data." In their classification of large-scale systems, Fleming and Von Hale characterize very large databases in terms of the number of rows in a table or the amount of data contained in a table.

A comparison of each author's description reveals distinct similarities exist in their perceptions of what constitutes a very large database. The foundation of each characterization rests upon the amount of data stored in tables within the database. Clearly, this factor plays a primary role in distinguishing a very large database system from smaller systems. Thus a database system can be classified as "very large" if it contains tables made up of several gigabytes of data. Gigabytes are used as a measure of table volume in this classification because they reflect the size of a row as well as the number of rows within a database table.

Examples of Very Large Databases

Common examples of computer systems handling very

large volumes of data can be found in the following domains.

Telecommunications industry. Utilize very large databases for recording local and long-distance telephone calls for millions of customers, as well as the tracking of billing and receivables information.

Insurance companies. Large-scale systems used for storing information about insurance policies, premiums, payments, losses and claims for millions of policy holders.

Banking institutions. Depicting information on millions of clients along with historical records of individual financial transactions in very large database systems.

Marketing agencies. Very large databases provide for the tracking of nationwide mailing list along with pamphlets and brochures sent to several million addresses.

Some examples of very large databases presently being used are GTEL, Georgia Institute of Technology's Library Automation System[4]. This database contains over 2,250,000 records which are updated on a daily basis. Another example is PLATO, Programed Logic Automation Teaching Operations, developed by the University of Illinois[5]. PLATO is a computer assisted instructional system that contains over 700 lessons on various subjects such as mathematics,

physics, chemistry, political science, etc. Also, SITA, Societe' Internationale de Telecommunications Aeronautiques, a message switching network, is a very large database[6]. SITA connects over 175 airline carriers and includes many on-line reservation computers.

Problems of Very Large Databases

Even though very large databases provide a vast amount of information and an array of services to its users, several problems still exist within these systems.

The phenomenal advancements made in computer technology over the past decade has resulted in reduced computing costs. Moreover, the price of computing has plunged by nearly a millionfold, yet the cost of implementing very large databases has not decreased proportionately. Additionally, the cost of implementing and maintaining a very large database system is extremely high. Thus we could accept the view so much is already invested in the current, large databases, that the majority of them will never be replaced by a completely new system, designed from scratch. With new hardware and software advancements in database technology being introduced on a frequent basis, the costs associated with these updates create a major expense problem for an enterprise.

A problem of very large databases that is of major

concern to this thesis is retrieval of data. The topic of retrieval has had, perhaps, the most influence in design and implementation of existing database systems. Retrieval methods are a vital factor to a database system because these methods have a major impact on system performance. Performance in a database system ultimately translates to one thing - response time. Response time is the amount of time a user has to wait from the moment a transaction is entered at a terminal until the first of the output from the transaction is returned. Naturally, this performance can even expect to worsen as the database grows larger thus forcing an increasingly finer level of resolution in this area.

Several additional problems occur in very large databases due to retrievals. Obviously, since a vast amount of data is stored in these systems, retrievals turn out to be a very time consuming act, thus making the query access time very slow with large databases.

A poorly designed database system can also create retrieval problems. Data retrieval can turn out to be slower if results have to come from several different tables rather than from a single table. On the other hand, larger tables can require the system to scan through more data than is absolutely necessary in order to retrieve the desired information. Clearly, the number and size of tables in a database directly affects retrieval time. So as the number and size of tables in

a system increases, retrieval times can expect to worsen.

The way in which a query is structured can also affect retrieval times. A query is simply a demand placed upon a database. Queries involving several tables can adversely affect retrievals since the data is dispersed among numerous tables. Once the needed tables are located, the system must join the tables and extract the required information. This process can take a considerable amount of time if several tables are involved in a query.

One method for speeding up data retrieval is the utilization of indexing[7]. An index is a mechanism for locating data stored in the database. Just as an index in a book allows the reader to quickly locate pages, an index on a column expedites data retrieval. Thus an index on a column can often make the difference between a rapid response to a query and a extremely long wait.

Another possible approach to boost retrieval time is the use of a query optimizer[8]. The query optimizer is responsible for decomposing every query into several segments and rearranging it to run as efficiently as possible. The query optimizer generates a strategy for retrieving the necessary data by seeking the most efficient path of retrieval.

The amount of memory allocated to the data cache

can also minimize retrieval time. The cache is the area of memory where the most recently used data is stored. By retaining commonly used data in the cache, retrieval of this data is without delay.

Overall, there are many challenges faced when attempting to overcome the deficiencies that exist within very large databases. If the system can understand the needs of the end-user, through intelligent and efficient design, constant retrieval of data could perhaps be avoided altogether. The future database system should try to answer as many questions about the needed database object as possible in order to obviate constant retrievals[9]. This kind of anticipation becomes even more critical in larger scale systems requiring real-time applications. Real-time systems are systems that respond fast enough so that the response can be used. Since such systems generally function based upon the timeliness of responses from a database, a utility for decreasing the often lengthy response times plaguing these systems, could dramatically enhance system performance.

Chapter 2

PROBLEM STATEMENT

A major area of concern with very large databases is that of query and access time. This topic has had, perhaps the most influence on design and implementation of current database systems. However, a major problem encountered in extremely large databases is that of retrieving data within small response times. Since a vast amount of data is stored in these systems, retrievals turn out to be a very time consuming act, thus making the query access time very slow. While quick response times are not vital to all database systems, many systems requiring real-time applications call for immediate results.

Based on the observed locality of reference in computing: When a database object is fetched, it has a high probability of being needed again and again. Thus constant retrieval of this database object can turn out to be a very time consuming task. It would be advantageous for the system to answer as many questions about the needed database object as possible in order to obviate constant retrievals. Consequently, a process for anticipating queries could play a major role in reducing the number of retrievals. While most Data Base Management System (DBMS) packages are designed to

provide both query flexibility and access speed, mechanisms for anticipating queries are not contained in these packages[10]. Nevertheless, in many instances predictability of queries for improved response time holds promise.

One area of interest is found in very large database systems where small response times are considered crucial to the end user. An example of this type of system would be found in a hospital environment where the medical staff ultilizes a very large database to make decisions on surgical procedures. A system which supports query prediction would allow the medical staff to make faster, live-saving decisions during surgical procedures since the needed data is already available. Another example of a real-time system can be found in a nuclear power plant where a very large database is used to make instantaneous decisions on power requirements for the plant. If the needed data is readily available to the plant management, key decisions can be made instantly, therefore keeping plant operation cost at acceptable levels.

Usually, when the end user poses a query, the query is received by the DBMS. In turn, the response to that particular query is returned to the user. This process continues until the user poses all desired queries. While this retrieval method yields the desired results, database operations involved are duplicated many times

within the session. To get some idea of the task involved in the worst case in processing such a query, suppose q1 (query1) takes time t1 to process. Then the second query q2 will take time t1+t2. Furthermore, query q3 will take time t1+t2+t3 and so on. If T(q) is a measure of time to process a query, then

$$T(q1) = t1$$

$$T(q2) = t1 + t2$$

$$T(q3) = t1 + t2 + t3$$

$$T(qn) = t1 + t2 + t3 + \ldots\ldots + tn-1 + tn$$

$$T(qn) = \sum_{i=1}^{n} ti$$

As illustrated by the above equation, processing of the nth query can be an extremely tedious task since it includes the processing time of all previous queries. Hence this method is not be feasible in real-time applications.

Given query q1, we wish to predict, pose and respond to queries q2, q3,.....,qn so that if any of the queries qi(i>1) are subsequently posed, the response will be readily available. Thus it is possible to obviate the long response times characterized above, thereby making it possible to implement many real-time systems. The problem of this thesis is formulated as follows.

To illustrate the effectiveness of a tool for predicting , posing, and responding to subsequent queries, consider the task involved in the processing of query qn (where qn represents the nth query as determined above). Since the responses to all subsequent queries q2, q3,....qn are generated after the initial query q1 is posed and response generated, the response to the nth query is available immediately.

This study on query prediction offers a method for anticipating and formulating responses to queries that are most likely to be posed to the database. By developing a query prediction method, response times will be greatly reduced since the required data will already available.

Chapter 3

METHODOLOGY

In scientific research environments, queries to the database are posed in an ordered fashion. That is, researchers continuously pose queries to the database regarding a particular database object in such a fashion that each query is often based upon the previous query. Consequently, each query serves as a "stepping stone" to subsequent queries. This ordered approach to posing queries motivates the need for a technique for anticipating queries within scientific databases.

Basic Strategy

The basic strategy for the formulation of a methodology for anticipating queries will focus upon two major areas: the schema of the database and queries to be posed to the database by users. By concentrating on these key areas, the groundwork for a query anticipation system is paved.

Components of Methodology

A. Database Schema

The first component of the methodology is based upon the database schema. Design of the conceptual database schema consists of a process of transforming the system specifications into a database design that

humans can understand[11]. The conceptual database design serves as a guide to database users by representing the "real world" via entity and relationship types. Entity types are objects or events about which the database contains information. Relationship types are the associations between entity types within the database. Hence this schema exemplifies a model for bridging the gap between reality and a computer system. The foundation for the conceptual used in this thesis is the entity-relationship (E-R) model, developed by Peter Chen in 1976[12]. The E-R diagram is a graphic representation of entity types and relationship types. The basic constructions in the E-R model are the familiar entities, attributes, and relationships, all of which are represented in E-R diagrams. At the most basic level, E-R modeling means identifying the important things - entities - about which information will be stored in the database system, identifying the important properties of these entities, and identifying the important relationships among them.

The sample database used in this thesis is a Real-time Energy Management System for a Power Plant (PPD). This real-time system uses a very large database. The example used in this study focuses upon the alarm-controller-generator subschema of the database. The E-R model for the PPD database is obtained by taking

the E-R approach to modeling. The steps to E-R modeling are as follows:

1. Specify Entity Types

Entity types are objects or events about which the enterprise wishes to collect data. In the E-R model, entities are drawn as rectangles. The entity types of the PPD database are shown in Figure 1.

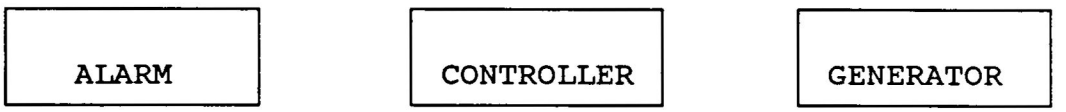

FIGURE 1 - ENTITIES FOR PPD DATABASE

Each of the above items is a entity type with an independent existence in the world of the Power Plant database. Each is represented in the database by a table.

2. Identify Primary Relationships Between Entities

A relationship is a correspondence or association between entities. Entities are related in the real world and the defined database structure should reflect these relationships. A relationship displays a logical, meaningful connection between two or more entity types.

A relationship between two (or more) entities implies that the entities are in some way associated with each other. Relationships are drawn as diamonds, with lines connected to the entities involved. Both entities and relationships are named in the E-R model. The lines are labeled to indicate the connectivity of the relationship. Connectivity describes the association between entity types. Between any two entity types there are three possible degrees of association: one-to-one (1:1), one-to-many (1:n), and many-to-many (m:n).

In the PPD database, the relationship between ALARM and CONTROLLER is 1:n. This relationship is diagrammatically illustrated in Figure 2.

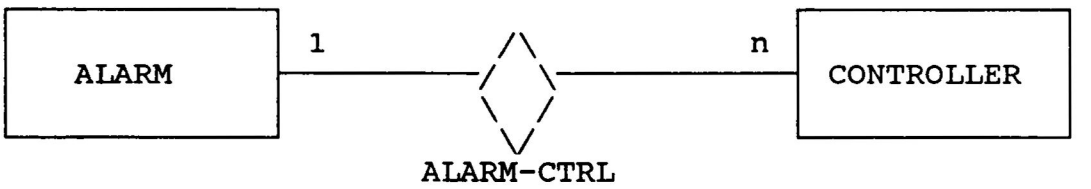

FIGURE 2 - ALARM-CTRL RELATIONSHIP TYPE

The ALARM-CTRL relationship type describes the association between alarms and controllers. This association means that at a given instant in time, a particular alarm is controlled by zero, one or many controllers. Conversely, each controller controls only

one alarm.

The GEN-CTRL relationship type, as shown in Figure 3, describes the association between generators controlled by each controller. The many-to-many association between the two entity types means that at a given instant in time, each controller controlls zero, one, or many generators. Also, each generator is controlled by zero, one, or many controllers.

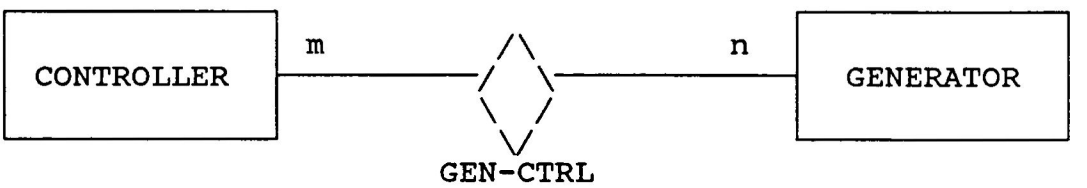

FIGURE 3 - GEN-CTRL RELATIONSHIP TYPE

3. Specify Attributes and Domains

Attributes are those properties of an entity type about which an enterprise wishes to collect information. Domains are the types of values an attribute can take (such as names, numbers etc.). Each individual entity type contains an attribute that uniquely distinguishes that particular entity from all others of that type. That "unique identifier" is called a primary key. For example, STUDENT# would normally be the primary key for

entity type STUDENT. Every student has their own unique social security number, and that number can be used to identify each student and to distinguish a particular student from all others.

The attributes/domains of the PPD database are listed in Figure 4.

ENTITY TYPES:	ALARM	CONTROLLER	GENERATOR
ATTRIBUTES :	alarm#	controller#	generator#
	type	model	level
	status	status	model
	location	location	location

Figure 4 - ATTRIBUTES/DOMAINS OF PPD DATABASE

The ALARM entity type consists of primary key alarm# identifying each entity type (alarm). The attribute alarm# is underlined to indicate it is the primary key for this entity type.

The CONTROLLER entity type consists of the primary key controller# identifying each controller entity type. Also, generator# is the primary key of the GENERATOR entity type.

4. The Global E-R Model

The conceptual design is achieved by connecting the E-R elements of the enterprise. By superimposing like entity types, the conceptual model is derived. See Figure 5.

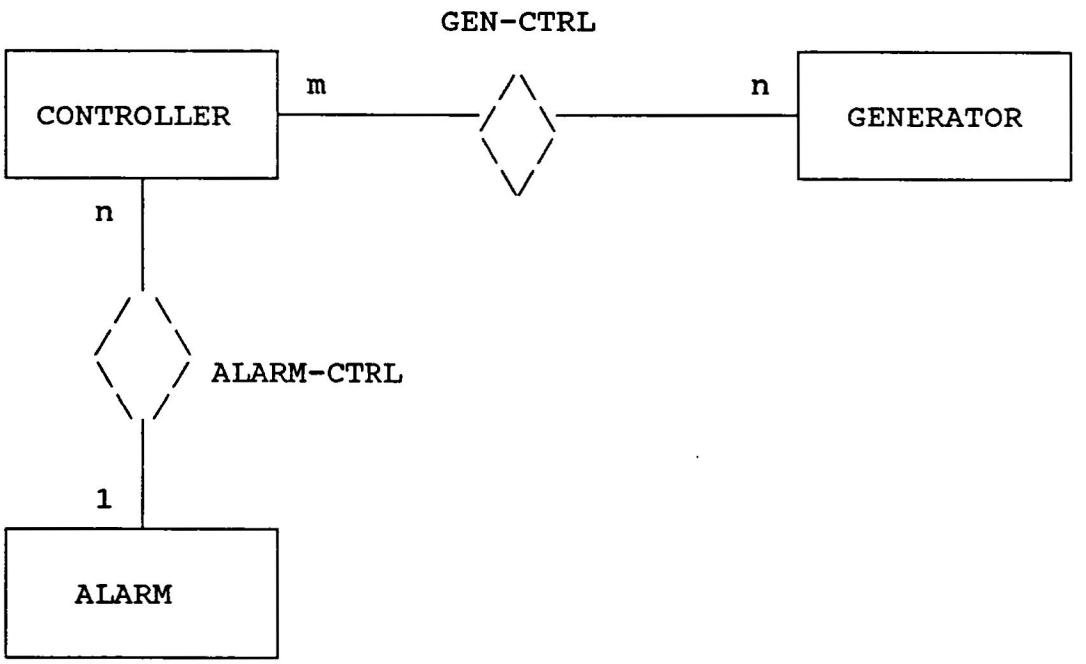

Figure 5 - CONCEPTUAL MODEL FOR PPD DATABASE

Once the E-R model for a given database is designed, this model can be converted into a corresponding logical schema design, as shown in Figure 6. The logical schema design serves as a guide which specifies the relations and relationships of the

database. A relation is a table of data values of the attributes of an entity type. The association between relations reflects a relationship. The E-R model for the PPD database contains three entity types, as shown in Figure 1, that can be represented in a DBMS as a table of data values, that is, relations. However, the relationships between the entity types are not represented in such a fashion. Therefore a means by which the E-R elements can be represented in a DBMS must be developed.

```
ALARM (alarm#, type, status, location)
CONTROLLER( controller#, model, status,
                location, alarm#)
GEN-CTRL( controller#, generator#, time)
GENERATOR( generator#, level, model, location)
```

Figure 6 - LOGICAL SCHEMA DESIGN FOR PPD DATABASE

Consider the 1:n relationship which connects the ALARM and CONTROLLER entity types. Both entity types in this association can be represented by relations. However, representation of the ALARM-CTRL relationship type is not accounted for. In order to represent ALARM-CTRL as a relation consider the situation shown in Figure 7. The diagram follows the 1:n association

between ALARM and CONTROLLER: For each alarm there may be several corresponding controllers, and many controllers may control one particular alarm.

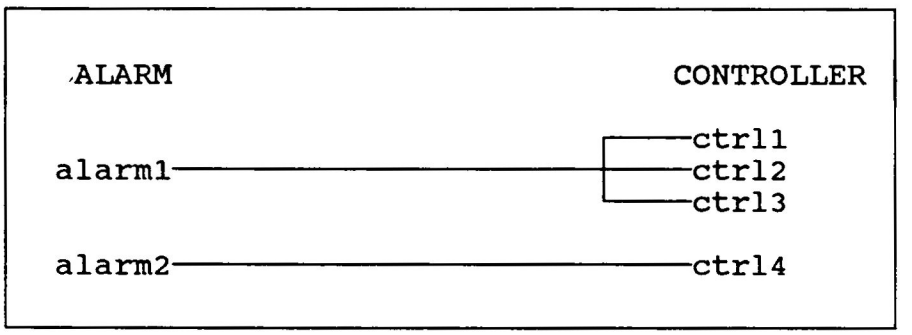

Figure 7 - MAPPING DIAGRAM FOR ALARM-CTRL

In Figure 7, ctrl1, ctrl2, ctrl3 belong to alarm1. This ownership can be expressed implicity by modifying the CONTROLLER relation so that it contains alarm#. In the modified relation, alarm# is a foreign key. Foreign keys are those attributes which serve a primary key in some relation within the same database. The example illustrates how 1:n relationships are represented in the database: The "many" side relation (CONTROLLER in this example) includes a foreign key whose values match values of the primary key of the "one" side relation (ALARM). The ALARM-CTRL relationship type is retained in this translation by the existence of the foreign key alarm# in the relation

CONTROLLER. Diagrammatically, this situation is shown in Figure 8.

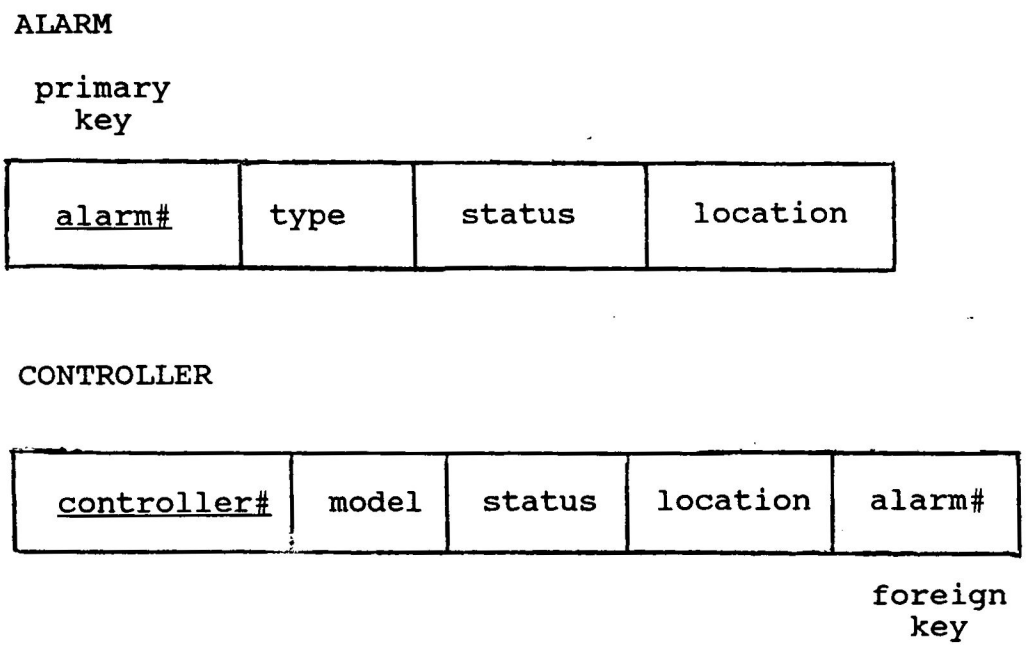

Figure 8 - MAPPING OF ALARM-CTRL RELATIONSHIP TYPE

The GEN-CTRL relationship type is a result of the many-to-many relationship between CONTROLLER and GENERATOR. Since both entity types contain a "many" side, a separate relation may be formed to represent the many-to-many relationship. This separate relation uses the foreign keys to identify the entity types involved in the relationship. The combination of the foreign keys serves as the primary key for the relation. Such a key is called a concatenated key. The relationship type

is preserved in the translation by the existence of the foreign key controller# in relation GEN-CTRL, and foreign key generator# in relation GEN-CTRL, as shown in Figure 9.

CONTROLLER

primary key

controller#	model	status	location

GEN-CTRL

foreign key

controller#	generator#	time

foreign key

GENERATOR

primary key

generator#	level	model	location

Figure 9 - MAPPING OF GEN-CTRL RELATIONSHIP TYPE

Primary key/foreign key matches are the glue that holds the database together. The special relationship between the primary key and the foreign key for a relationship type, permits two or more relations in the database to be joined.

B. User Queries

The second component to the methodology is the

examination of user queries. A query is simply a demand placed upon a database to retrieve, update, insert, or delete data. There is a correspondence between user queries and the database schema. This correspondence classifies certain patterns of English usage into a corresponding database schema. In order to identify the association between user queries and database schema, guidelines for translating an English statement into an E-R diagram must be followed. Several rules were developed by P. Chen for such translations[13]. The translation rules of interest to this thesis are as follows:

Rule 1. A common noun (such as "student," "department") in English corresponds to an entity type in an E-R diagram.

Rule 2. A transitive verb in English corresponds to a relationship type in an E-R diagram.

Example A:
English statement - " An instructor teaches a student and belongs to a department. "
Analysis: Note that "instructor," "student," and "department" are nouns and therefore serve as entity types. Note also the "teaches" and "belongs to" are transitive verbs (or verb phrases) and therefore

correspond to relationship types. The corresponding E-R diagram is shown in Figure 10.

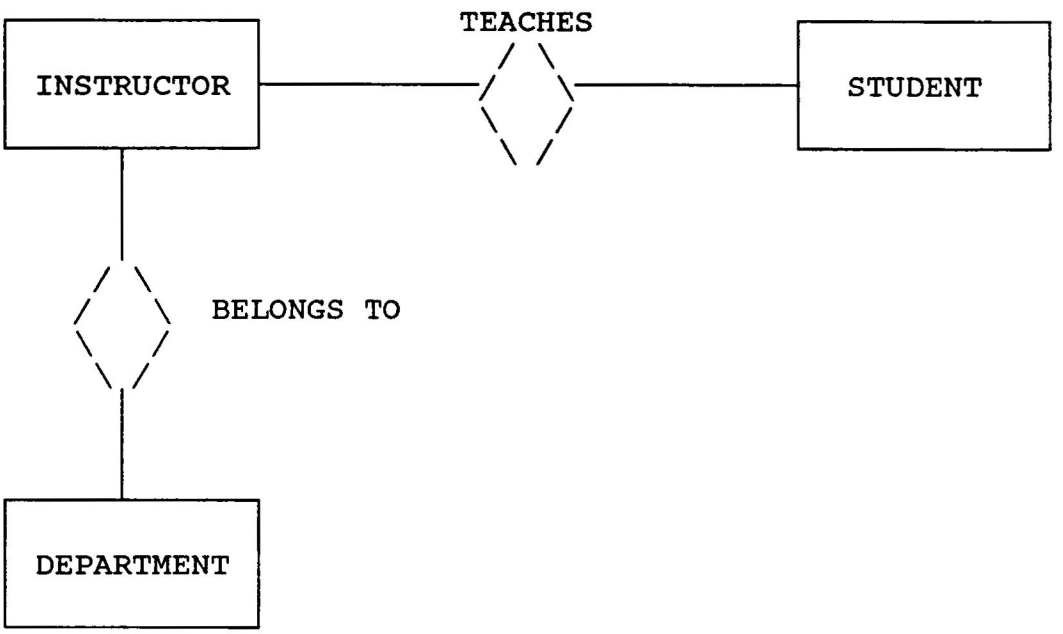

Figure 10 - E-R Diagram for Example A

Rule 3. An adjective in English corresponds to an attribute of an entity in an E-R diagram.

Example B:
English statement - " Instructor #15 teaches Math 101."
Analysis: "Instructor" and "course" are nouns and can be considered as entity types. Since "#15" is an adjective describing a particular instructor, instructor# can serve as an attribute for entity type INSTRUCTOR. Also, " Math 101 " is an adjective

modifying the noun "course." Thus course# serves as an attribute for entity type COURSE. Finally, "teaches" is a transitive verb and therefore serves as a relationship type.

Clearly there is a correspondence between user queries and the underlying logical database schema. Thus given an environment in which a user repeatedly queries for a particular database object, knowledge of the relations contained in the initial query will lead directly to the portion of the database that will likely be queried next. Starting from the initial query, the database relations may be navigated along the primary key/foreign key associations in a way that predicts queries. In the following we outline the steps for anticipating queries based upon the above motivation.

Steps for Anticipating Queries

When a user query is posed to a database, the method for anticipating future queries is as follows:

STEP 1: The database system examines the initial query and generates a response.

STEP 2: The initial query is analyzed to determine the relations involved in it.

STEP 3: The database schema is analyzed to establish those relations associated with the relations of the initial query.

STEP 4: Relationships between the relations in the initial query and associated database relations (as determined in Step 3) are examined to obtain a set of generalized anticipated potential queries.

STEP 5: Responses to the generalized queries formed in Step 4 are obtained by submitting them to the DBMS.

Application of Methododology to PPP Database

Now consider the PPD database schema illustrated in Figure 5. This illustration is the E-R model for the given database developed during the conceptual phase of database modeling. By mapping this model into a logical design, Figure 6 is obtained. These diagrams represent the first component of the methodology.

The second component of the methodology is the evaluation of a given user query. Suppose a database user wishes to gather miscellaneous information concerning relation ALARM, then an initial query

must be formulated in order to probe the database so this data can be extracted. Queries posed in this methodology will be constructed using SQL (Structured Query Language), which is the standard relational language[14]. However, the methodology is not restricted to SQL. Any relational language could be applied.

Applying the query anticipation tool to the database yields the following:

Step1: Query#1: Obtain miscellaneous information about Alarm# 1.

The SQL statement for this query is as follows:

```
select  *
from    ALARM A
where   A.alarm# = '1'
```

This query (q1) is posed to the database and a response is generated.

Step2: Those relations utilized in a given query in the 'from' clause within the SQL statement are the relations involved in the query. Evaluation of this query

shows that ALARM is the only relation involved in this initial query.

Step3: The logical design is analyzed to establish those relations associated with ALARM. Traversing the diagram via the foreign keys, results in ALARM being associated with CONTROLLER via foreign key alarm#. Furthermore, CONTROLLER is associated with GEN-CTRL via controller#. Also, GEN-CTRL is associated with GENERATOR via foreign key generator#.

Step4: The relationship between ALARM(the relation in the initial query) and CONTROLLER(the associated relation) is examined in order to formulate a general query type. This query type exploits the relationship between ALARM and CONTROLLER. Additionally, the relationships between CONTROLLER and GEN-CTRL, and GENERATOR and GEN-CTRL are evaluated in order to formulate query types.

The queries generated by this traversal of the logical design are:

Query type: ALARM - CONTROLLER
Obtain information about controllers controlling alarm#1.

Query type: CONTROLLER - GEN-CTRL
Obtain information about amount of time controllers associated with alarm#1 spend controlling generators.

Query type: GEN-CTRL - GENERATOR
Obtain general information about generators associated with alarm#.

Step5: The SQL statement for each query type is as follows:

Query type: Obtain information about controllers controlling alarm#1.

Corresponding SQL statement:

```
q2:     select  C.*
        from    CONTROLLER C
        where   C.alarm# = '1'
```

Query type: Obtain information about amount of time controllers associated with alarm#1 spend controlling generators.

Corresponding SQL statement:

```
q3:     select  T.*
        from    ALARM A, CONTROLLER C
                GEN-CTRL T
        where   A.alarm#=C.alarm#
                and C.crtl#=T.crtl#
                and A.alarm# = '1'
```

Query type: Obtain general information about generators assosciated with alarm#1.

Corresponding SQL statement:

```
q4:     select  G.*
        from    ALARM A, CONTROLLER C
                GEN-CTRL T, GENERATOR G
        where   A.alarm#=C.alarm#
                and C.crtl#=T.ctrl#
                and T.gen#=G.gen#
                and A.alarm# = '1'
```

The query prediction subsystem submits these SQL statements are to the DBMS in order to obtain the responses. Note that the responses to each query is in the form of a relation since the resultant formation is a table of values.

Implementation of PPD Database

The PPD Database was implemented on a VAX/VMS mainframe using INGRES[15]. INGRES is a relational DBMS well suited for a wide variety of applications. For example, an application program written in a high level language may access an INGRES database, while non-programmers who are skilled database users can meet their information management needs by using SQL. The application program for the PPD Database is written in "C" programming language[16]. Refer to Appendix B for a copy of the program.

The query anticipation tool developed in this thesis is executed in the application program by traversing a list of the primary keys/foreign keys of relations within the PPD Database. The primary key and the corresponding foreign key for each relation in the database are represented in a list. For those

relations containing no foreign key (such as the ALARM relation or GENERATOR relation), the primary key is listed in both the primary key and the foreign key fields. Thus "alarm" serves as the primary as well the foreign key in the list for relation ALARM. "Ctrl" and "alarm" are the primary key and foreign key respectively for relation CONTROLLER. "Gen" and "ctrl" are the primary key and foreign key representing the GEN_CTRL relation. Finally, "gen" represents the primary as well as the foreign key for the GENERATOR relation. See FIgure 11 for an illustration of the primary/foreign key listing utilized in the application program.

KEYS	(represents)	RELATION
alarm/alarm		ALARM
alarm/ctrl		CONTROLLER
ctrl/gen		GEN_CTRL
gen/gen		GENERATOR

Figure 11 - Attribute Listing for Application Program

The application program executes the query anticipation tool as follows:

STEP 1: Program contains a function (process_q1) that processes the initial query (q1). This query involves obtaining general information about alarm#1, and therefore involves the ALARM relation. The corresponding SQL statement for this first query is submitted to the DBMS and the results returned.

The results of Query 1 are:

alarm#	type	status	location
1	A	on	rm1

STEP 2: A function (lookup) is called that searches through the attribute listing for primary key/foreign key matches. Starting with the primary key of the initial relation ("alarm"), the list is traversed to find this same attribute in the foreign key field of some other relation in the list. A search for attribute "alarm" results in a match with

the foreign key field of the CONTROLLER relation. Once this primary key/foreign key match is found, a function (process_q2) that processes a query involving the ALARM and CONTROLLER relation that generates specific information about alarm#1 is called. Further traversal of the list is conducted by searching for the primary key of the CONTROLLER relation ("ctrl") in the foreign key slot within some other relation in the listing. A match is found with the GEN_CTRL relation and the corresponding program function (process_q3) is called in order to process the CONTROLLER - GEN_CTRL query type. Another traversal of the list is done in order to find the a match between the "gen" key in the GEN_CTRL relation with another relation in the database. A search of the GENERATOR relation results in a match via the "gen" key. Therefore a query can be processed for the GEN_CTRL - GENERATOR query type.

STEP 3: Each query type is processed and

responses generated:

Results to query#2:

ctrl_no	model	status	location	alarm_no
1	H	on	rm1	1

Results to query#3:

ctrl_no	gen_no	time
1	1	5

Results to query#4:

gen_no	level	model	location
1	10	A	rm1

The above responses are predicted queries and are available for immediate use as promised in the problem statement.

Chapter 4

CONCLUSION AND RECOMMENDATIONS

Conclusion

The methodology developed here offers a system that will not only respond to a given query, but also predict and respond to potential queries. This system anticipates queries by determining a pattern of potential queries based upon a given query and the database schema. Clearly, this method provides a very powerful tool for database systems requiring real-time applications since responses to anticipated queries are readily available.

Recommendations

Additional research in the following areas are recommended to further enhance the viability of the query prediction tool developed in this study.

1. Further investigation should be geared towards the development of a method for handling the excessive amounts of data maintained in temporary storage due to the multitude of tables generated by the prediction tool. Special consideration might be given to the issue of how to manage unwanted data or data the user no longer needs.

2. Further research should focus upon the modification of the query prediction tool so that this methodology could be applied to more complex database systems. The sample database introduced in this study was offered in the most general terms. However, in the real world, the majority of database systems are extremely complex and contain multiple relationships. Future work in this area should focus upon expanding the tool so that more complex database systems are accommodated.

APPENDIX A

PPD DATABASE

ALARM

alarm#	type	status	location
1	A	on	rm1
2	B	on	rm2
3	B	on	rm3
4	C	off	rm4
5	C	off	rm5

CONTROLLER

ctrl#	model	status	location	alarm#
1	H	on	rm1	1
2	I	on	rm2	2
3	J	on	rm3	3
4	K	on	rm4	4
5	L	off	rm5	5

GEN_CTRL

ctrl#	gen#	time
1	1	5
2	2	5

GENERATOR

gen#	level	model	location
1	10	A	rm1
2	10	B	rm2
3	10	C	rm3
4	10	D	rm4
5	10	E	rm5

APPENDIX B

```
/*  include  <stdio.h>  */


/*  Query Anticipation Program   */


exec sql include sqlca;
exec sql begin declare section;
exec sql include "alarm.dcl";
exec sql include "ctrl.dcl";
exec sql include "genctrl.dcl";
exec sql include "gen.dcl";
exec sql end declare section;


init_db()

{

exec sql whenever sqlerror stop;
exec sql connect sample;

}


process_q1()      /*  function to process query1  */

{

     exec sql whenever sqlerror call close_down;
     exec sql select *
          into       :almrec
          from   alarm
          where  alarm_no  =  '1';



     printf("           query#1 results           \n");
     printf("-------------------------------------\n");
     printf("\n");
     printf(" %5s, %5s, %5s, %5s  ", almrec);
     printf("\n");
}
```

```
process_q2()    /*  function to process query2  */

{

     exec sql whenever sqlerror call close_down;
     exec sql select *
          into       :ctrlrec
          from       controller
          where      alarm_no = '1';



     printf("           results of query#2       \n");
     printf("------------------------------------\n");
     printf("\n");
     printf(" %5s, %5s, %5s, %5s, %5s ", ctrlrec);
     printf("\n");

}



process_q3()    /*  function to process query#3  */

{

     exec sql whenever sqlerror call close_down;
     exec sql select t.ctrl_no, t.gen_no, t.time
          into       :gcrec.ctrl_no, :gcrec.gen_no,
                     :gcrec.time
          from       alarm a, controller c, gen_ctrl t
          where      a.alarm_no=c.alarm_no
                     and c.ctrl_no=t.ctrl_no
                     and alarm_no = '1';



     printf("           results of query#3       \n");
     printf("------------------------------------\n");
     printf("\n");
     printf(" %5s, %5s, %5s ", gcrec);
     printf("\n");

}
```

```
process_q4()    /*  function to process query#4  */

{

     exec sql whenever sqlerror call close_down;
     exec sql select g.gen_no, g.level, g.model,
                    g.location
          into      :genrec.gen_no, :genrec.level,
                    :genrec.model, :genrec.location
          from      alarm a, controller c, gen_ctrl t
                    generator g
          where     a.alarm_no=c.alarm_no
                    and c.ctrl_no=t.ctrl_no
                    and t.gen_no=g.gen_no
                    and alarm_no = '1';



     printf("          results of query#4       \n");
     printf("-----------------------------------\n");
     printf("\n");
     printf(" %5s, %5s, %5s, %5s ", genrec);
     printf("\n");

}



end_db()

{
exec sql disconnect;
}



close_down()

{
exec sql begin declare section;
char errbuf[101];
exec sql end declare section;
exec sql whenever sqlerror continue;
exec sql copy sqlerror into: errbuf with 100;

printf("closing down because of database error:\n");
printf("%s\n",errbuf);

exec sql abort;
exec sql disconnect;
exit(-1);

}
```

```
struct entry

{

     char   for_key[10];
     char   prim_key[10];

};



/*  function to determine if strings are equal  */

int equal_strings(s1,s2)
char s1[],s2[];
{

     int i=0,answer;

     while (s1[i] == s2[i] && s1[i] != '\0'
               && s2[i] != '\0')
       ++i;

     if (s1[i] == '\0' && s2[i] == '\0')
       answer=1;   /* strings equal */
     else
       answer=0;   /* strings not equal */

     return(answer);

}



/*  function to lookup a key inside attribute listing */

int lookup(list,search,no_of_entries)
struct entry list[];
char   search[];
int    no_of_entries;
{
     int i;

     for(i=0; i<no_of_entries; ++i)
       if(equal_string(search, list[i].prim_key))
         return(i);

     return(-1);

}
```

```
main()
{

     static struct entry list[4]=
        {  { "alarm",  "alarm"  },
           { "alarm",  "ctrl"   },
           { "ctrl",   "gen"    },
           { "gen",    "gen"    }  };


int  entries = 4;
char *word = "alarm_no";
int  entry_no;

entry_no = lookup(list, word, entries);
*word =  list[entry_no].for_key

{

 int  i;
 for (i = 0; i < entries, ++i)
 lookup(list, list[entry_no.for_key, entries)

 if (entry_no == 1)
  {
  init_db();
  process_q1();
  end_db();
  }
 else if (entry_no == 2)
  {
  init_db();
  process_q2();
  end_db();
  }
 else if (entry_no == 3)
  {
  init_db();
  process_q3();
  end_db();
  }
 else
  {
  init_db();
  process_q4();
  end_db();
  }

}
```

REFERENCES

[1]. Lockemann, P.C., and Neuhold, E.J., "Systems for large databases," Proceedings of the 2nd International Conference on Very Large Data Bases, North-Holland, Amsterdam, 1977.

[2]. Inmon, W.H., "Effective Data Base Design," Prentice-Hall, Englewood Cliffs, NJ, 1981.

[3]. Fleming, C., and Von Hale, B., "Handbook of Relational Database Design," Addison-Wesley, Reading, MA, 1989.

[4]. Georgia Institute of Technology, "Information for Library Visitors," Atlanta, GA, 1989.

[5]. Control Data Education Co., "PLATO Courses: 1980 Catalogue," Champaign, IL, 1980.

[6]. Schwartz, M., "Data Communication Network Design and Analysis," Prentice-Hall, Englewood Cliffs, NJ, 1977.

[7]. Emerson, S.L., "The Practical SQL Handbook: Using Structured Query Language," Addison-Wesley, Reading, MA, 1989

[8]. McFadden, F.R., and Hoffer J.A., "Data Base Management," Benjamin/Cummings Publishing Co., Menlo Park, CA, 1985.

[9]. Chen, T.C., "Computer Technology and the Database User," Proceedings of the Fourth International Conference on Very Large Data Bases, North-Holland, Amsterdam, 1978,

[10]. Relational Technology Inc., "INGRES/Embedded SQL User's Guide and Reference Manual," Alameda, CA, 1986.

[11]. Pratt, P.J., and Adamski, J.J., "Database Systems: Management and Design," Boyd and Fraser Publishing Co., Boston, MA, 1987.

[12]. Chen, P.P., "The Entity-Relationship Model: Towards a Unified View of Data," ACM Transaction on Database Systems, Vol 1, No. 1, March 1976.

[13]. Chen, P.P., "English Sentence Structure and Entity-Relationship Diagrams," Entity-Relationship Approach to Information Modeling and Analysis, North-Holland, Amsterdam, 1983.

[14]. Date, C.J., "A Guide to the SQL Standard," Addison-Wesley, Reading, MA, 1987.

[15]. Relational Technology Inc., "INGRES/Embedded SQL Companion Guide for C," Alameda, CA, 1986.

[16]. Kochan, S.G., "Programming in C," Hayden Books, Indianapolis, IN, 1988.